

1 [0001] SECURE, REAL-TIME APPLICATION
2 EXECUTION CONTROL SYSTEM AND METHODS
3
4

5 [0002] Inventors:
6 Duc Pham
7 Tien Le Nguyen
8 Pu Paul Zhang
9 Mingchen Lo
10
11
12
13

14 [0003] Background of the Invention
15 [0004] Field of the Invention:
16 [0005] The present invention is generally related to the establishment
17 of secure, fine-grained trust relationships between computer systems in multi-tier
18 distributed computing environments and, in particular, to a system and methods
19 of securely establishing the operative chain of trust down to the level of individual
20 application program instances as loaded in real-time for execution on host
21 computer systems.
22

23 [0006] Description of the Related Art:
24 [0007] Distributed computing environments depend on mutually
25 recognized trust relations among networked computer systems to establish
26 consistent control over the access and utilization of shared resources.
27 Conventional computer operating systems establish trust relations based simply

1 on a shared confidence in the identity of users. Various known network security
2 systems effectively enable a password authenticated user identity to be established
3 within a defined network space, such as the domain controller architecture initially
4 implemented in the Microsoft® WindowsNT® operating system and the various
5 yellow-pages services implemented in variants of the Unix® operating system.
6 Access control lists (ACLs) and similar user/group attributes established locally
7 against particular computer resources then control whether any particular user is
8 able to access and use a network resource.

9 [0008] Distributed computing environments have greatly increased in
10 complexity as required to meet ever widening operational demands that arise
11 from various topographical, commercial, and regulatory requirements. Since
12 networked computer systems can be highly decentralized, the network security
13 system must, as a practical matter, permit aggregated control to be delegated to
14 and performed by a centralized security administrator. Commercial requirements
15 for functionality, performance, and redundancy have driven adoption of multi-
16 tiered server computing environments, employing distributed application and
17 database servers, which require a chain of trust to be established across each
18 tiered level. Recent regulatory requirements have increased the need to assure
19 security over privacy related data and, further, provide an audit of access and
20 delivery of the data. Consequently, a need to significantly improve the security
21 throughout distributed computing environments and ensure the integrity of trust
22 relations formed between computer systems exists.

23 [0009] Various efforts have been made to improve distributed security
24 systems as an essential step toward establishing and maintaining distributed trust
25 relations. These efforts include, among others, controlling access to specific

1 resources by applications and other executables and securing network
2 communications between executing applications. By controlling and, as
3 appropriate, restricting access to certain computer resources, both untrusted and
4 trusted but misused applications are prevented from abusing the pre-established
5 trust relationship between the user and the computer system and, further, between
6 computer systems within a distributed computing environment.

7 [0010] Restricted application access control systems typically build on
8 existing password authenticated user identity systems in an attempt to securely
9 manage the execution of specific application programs. For example, Fischer (US
10 Patent 5,412,717) and Chan et al. (US Patent 6,505,300) each describe restricted
11 execution environments implemented integral to the local operating system. The
12 Fischer system conditions the execution of an application or other executable
13 content within the restricted environment on local verification of a secure
14 application signature. Known, unmodified applications are then permitted to
15 execute subject to assigned constraints on the resources that can be accessed by
16 the application. A constraint profile, which is locally associated with an
17 application based on the identity of the application or application class, is used
18 by the restricted execution environment to filter each attempt by an executing
19 application to access a resource. Only accesses explicitly permitted are allowed
20 to proceed. The Chan et al. system adds a fairly complex access control list
21 capability to the constraint profile, thereby increasing the fine-grained
22 specification of whether different resources, including other executing programs,
23 may be accessed by an executing application.

24 [0011] Even where applications, as executed, are entirely well-behaved,
25 maintaining a trust relationship across a distributed computing environment

1 requires all communications between applications to be maintained secure
2 against electronic attack, including interception, redirection, and eavesdropping.
3 Secure communications are typically achieved by encrypting transmitted data,
4 typically using a form of public key encryption. Secure communications channels
5 are established in a variety of ways. Secure communications services can be
6 added directly to the network operating system environment to support virtual
7 private networks (VPNs). Typically, VPN communications systems provide a secure
8 communications channel established between disparately located computer
9 systems.

10 [0012] While preventing external attack, conventional VPNs are shared
11 services that permit applications executing on either end-point computer system
12 to use the communications channel, thereby remaining open to attack from other
13 users and applications executing on the end-point systems. To reduce exposure
14 to internal attacks, various approaches have been advanced to establish and
15 control multiple, discretely encrypted VPN channels between the same end-point
16 systems. For example, multiple virtual routers, each representing a separate VPN
17 channel, can be established at each end-point. Ylonen et al. (US Patent
18 6,438,612) describes a multiple, virtual router system that supports independent
19 encryption of each virtual router channel. Use of any particular router channel is
20 determined by presentation of a uniquely corresponding virtual network identifier
21 representing, effectively, an extended IP address. Multiple applications and other
22 executable content assigned the same virtual network identifier, presumptively on
23 the basis of equal trustworthiness, will use the same virtual router channel.
24 Unfortunately, while increasing the number of VPNs available for use, internal

1 attacks need only spoof a targeted virtual network identifier in order to gain
2 access to communications between otherwise secured applications.

3 [0013] An alternate approach is to establish secure execution environments
4 that internally provide for secure network communications. Conventionally,
5 secure shell (SSH) containers are selectively executed on end-point computer
6 systems as alternatives to the native shell execution environments provided by the
7 host operating systems. Each secure shell, in turn, supports an execution context
8 that enables execution of one or more contained or hosted applications. Network
9 communications between independently hosted applications are filtered through
10 and fully encrypted by the mutual operation of the secure shells. Thus, while the
11 secure shells support a relatively more controlled environment for executing
12 applications that could securely share a single communications channel, there are
13 substantial complexity and security management issues inherent in reliably
14 configuring multiple secure shell environments on multiple, disparately located
15 computer systems. In addition, any internal attack that permits a compromised
16 application to be executed as a secure shell hosted application is then able to gain
17 access to the otherwise secure communications of the other commonly hosted
18 applications.

19 [0014] Another conventional approach to ensuring secure communications
20 between individual applications is to directly implement a security protocol, such
21 as the secure sockets layer (SSL) protocol, as an integral part of the application
22 itself. Conventionally, communicating applications must be specifically written to
23 interact with and utilize the functions of the secure sockets layer implemented at
24 each end of an otherwise shared communications channel. The available security
25 functions, such as the ability to require certificate authentication of the

1 participating applications, is, however, limited to the SSL API revision level
2 commonly supported by the communicating applications.

3 [0015] While the SSL and, to varying extents, other application-level security
4 protocols are accepted and used, there are inherent drawbacks to their use. Each
5 application must be not only initially written to use a specific security protocol, but
6 frequently revised to maintain compatibility with and support the functions
7 available in later revisions of the protocol API. Furthermore, the available security
8 operations are limited to the established set of procedures included in the security
9 protocol specification. Protocol extensions to establish and enforce additional
10 qualifications on the use of a secured channel, as may be appropriate in specific
11 business processes, are generally not possible. Such extensions would have to be
12 implemented as part of proprietary application programs and would therefore
13 interoperate only between those applications.

14 [0016] Consequently, there is a distinct need for a secure mechanism
15 capable of establishing trust relationships on and between computer systems
16 including particularly to found the trust relationship at the point of loading
17 applications for execution at any tier within a multi-tier distributed computing
18 environment.

19

1 [0017] Summary of the Invention

2 [0018] Thus, a general purpose of the present invention is to provide a
3 system and methods of enabling a chain of trust to be established to individual
4 application program instances as loaded as loaded in real-time for execution on
5 host computer systems.

6 [0019] This is achieved in the present invention by providing a security
7 server to qualify the execution of programs on networked host computer systems.
8 The security server uses a database that stores pre-qualified program signatures
9 and defined policy rules associating execution permission qualifiers with execution
10 control values. The server executes a control program in response to execution
11 requests received via a communications network interface from identifiable hosts,
12 wherein a predetermined execution request received from a predetermined host
13 computer system includes an identification of a program load request, request
14 context related data, and a secure program signature. The control program
15 determines an execution control value based on an evaluation of the execution
16 request relative to the pre-qualified program signatures and defined policy rules.
17 The execution control value is then returned to the predetermined host computer
18 system to securely qualify the execution of the program identified from the
19 program load request.

20 [0020] Thus, an advantage of the present invention is that a chain of trust
21 can be established for individual processes by securely qualifying, in real-time,
22 each individual program instance as loaded for execution. Based on
23 administratively established policy rules and administratively pre-qualified secure
24 program signatures evaluated in connection with the loading of an application
25 image, the execution of the application can be securely qualified and explicitly

1 denied, permitted, or permitted subject to policy rule specified execution time
2 qualifications.

3 [0021] Another advantage of the present invention is that each program
4 instance can be evaluated in the real-time process of being loaded by an external
5 security server. A local policy enforcement module implemented as a component
6 of the operating system permits intercept of all operating system calls that could
7 result in the execution of a program and submits the load request for qualification
8 by a network connected and therefore independently secured security server.

9 [0022] A further advantage of the present invention is that the security
10 server qualifies the execution of programs for a well-defined community of host
11 computer systems, thereby enabling trust relations to be established for individual
12 application instances relative to their host computer system and, further, as a
13 foundation for establishing trust relations between application instances executing
14 in different host computers.

15 [0023] Still another advantage of the present invention is that the full
16 capability provided by the evaluation of policy set rules is available to qualify and
17 further constrain execution of program instances. The context associated with any
18 request to load and execution a program is made available for selection of
19 controlling policy set rules. Additionally, a secure signature of the program image
20 requested for execution is also provided to control rule selection. Provision of the
21 secure signature allows a path independent and therefore more secure and
22 universal identification of the specific program requested for execution. Rule
23 matching can therefore be extremely fine-grained, which provides substantial
24 administrative flexibility.

1 [0024] Yet another advantage of the present invention is that the product
2 of policy set rule evaluation can provide multiple possible determinations.
3 Execution of a particular program instance can be specified as a result of rule
4 evaluation to deny, permit, or permit subject to specified constraints. Applicable
5 constraints can be specified to the same fine-grained level applicable to the
6 matching of any of the policy set rules. Applicable constraints can define
7 administrative limitations, such as logging levels and auditing alarms, and
8 procedural limitations, such as execution permitted for only limited periods, at
9 only limited times, or subject to controls on the data or other system resources
10 otherwise available.

11

1 [0025] Brief Description of the Drawings

2 [0026] Figure 1 is a generalized view of a preferred operating environment

3 for a preferred embodiment of the present invention;

4 [0027] Figure 2 is a detailed view of a preferred operating interrelationship

5 between host computer systems in accordance with a preferred embodiment of

6 the present invention;

7 [0028] Figure 3 is a generalized diagram of a security server computer

8 system constructed in accordance with a preferred embodiment of the present

9 invention;

10 [0029] Figure 4 is a block diagram of the preferred data structure

11 organization of signature, reference group and policy databases as implemented

12 in a preferred embodiment of the present invention;

13 [0030] Figure 5 is a flow chart showing the preferred processing of

14 intercepted operations requests by a security server computer system in

15 accordance with a preferred embodiment of the present invention;

16 [0031] Figure 6 provides a generalized block diagram of a host computer

17 including a preferred software architecture implementing a policy enforcement

18 module in accordance with a preferred embodiment of the present invention;

19 [0032] Figure 7 is a software block diagram of an implementation of a

20 policy enforcement module within the kernel space of an operating system in

21 accordance with a preferred embodiment of the present invention;

22 [0033] Figure 8 is a flow chart illustrating a preferred failover operation of

23 the policy enforcement module in performing host-based encryption in

24 accordance with a preferred embodiment of the present invention;

1 [0034] Figures 9A-B are block diagrams illustrating multiple modes of
2 operation including local and remote encryption, compression, and tunnel
3 routing in accordance with a preferred embodiment of the present invention;

4 [0035] Figure 10 is a flow chart illustrating the opening of an application
5 instance in accordance with a preferred embodiment of the present invention;

6 [0036] Figure 11 is a flowchart illustrating the opening of an encrypted
7 communications channel in accordance with a preferred embodiment of the
8 present invention;

9 [0037] Figure 12 is a flowchart illustrating the operation of an encrypted
10 communications channel in accordance with a preferred embodiment of the
11 present invention; and

12 [0038] Figure 13 is a flowchart illustrating the closure of an encrypted
13 communications channel in accordance with a preferred embodiment of the
14 present invention.

15

16

1 [0039] Detailed Description of the Invention

2 [0040] The present invention enables fine-grained trust relationships to be
3 securely established for individual application instances, which is applicable both
4 to discretely qualify the execution of individual application instances and, further,
5 qualify and secure communications between individual application instances as
6 executed typically on network connected host computer systems. In the following
7 detailed description of the invention like reference numerals are used to designate
8 like parts depicted in one or more of the figures.

9 [0041] Figure 1 illustrates a variety of the configurations 10 supported by
10 the present invention. In general, the present invention enables specific
11 operations of the local operating system of a host computer system to be qualified
12 against an external database of security rules that define the permitted actions of
13 a fine-grained security policy for a computer domain subscribed to a security
14 server computer system. The qualified operations preferably include the loading
15 of application instances for execution and the establishment of communications
16 channels between individual application instances as executed on one or more of
17 the domain host computer systems. In the preferred embodiments of the present
18 invention, the security server computer system may be implemented as one or
19 more security appliances that may be physically sited locally or remotely with
20 respect to the various host computer systems.

21 [0042] A typical network configuration 10 employing the present invention,
22 as generally shown in Figure 1, provides for the secure qualification of tiered
23 interoperating application instances. In this configuration, a host computer 12
24 executes a local instance of an application loaded from local or remote storage
25 in a defined process context. Initial execution of the application instance is

1 authorized and authenticated relative to the process context. This local
2 application instance establishes a securely qualified communication channel with
3 another similarly authorized and authenticated application instance, executed on
4 an application server 14 to access, through a database server 16, data stored in
5 a database 18. The data transferred between the application server 14 and
6 database 18 is preferably protected through encryption operations implemented
7 by a core security appliance 20 as described in Secure Network File Access
8 Control System, by Pham et al. (Application SN 10/201,406; filed July 22, 2002;
9 now US Patent 6,678,828), which is incorporated by reference herein.

10 [0043] Communications channels, such as the channel between host
11 computer system 12 and application server 14, are established under the secure
12 control of a security appliance 22 operating through locally installed policy
13 enforcement modules (PEMs) 24, 26. The security appliances 20, 22 may be
14 physically discrete units configured for specific roles or, preferably, configured to
15 support multiple roles as needed by the same physical unit. Even where a security
16 appliance 20, 22 can support multiple roles, additional security appliances 28
17 can be employed to permit flexibility in the siting of physical devices, such as
18 where a host computer system 30, including locally installed PEM 34, is distant
19 from a security appliance 22 so as to be preferentially associated with a separate
20 security appliance 28.

21 [0044] As illustrated in Figure 2, a security appliance 42 is employed to
22 securely qualify local operations of application instances executed on host
23 computer systems 44, 46 through the operation of PEMs 48, 50, which are locally
24 installed and executed on the host computer systems 44, 46. Filesystem accesses,
25 such as to a direct attached store 52 or other stores accessible through the

1 network 54, can be qualified down to the level of individual application instances.
2 The PEMs 48, 50 further permit qualification of communications between the host
3 computer systems 44, 46 at any desired trust relation level down to the level of
4 individual application instances. In the preferred embodiments of the present
5 invention, PEMs 48, 50 are configured to intercept certain local and network
6 control and data access operations initiated by local application instances, such
7 as application instance 56, as well as remotely initiated access operations that are
8 directed to the application instance 56 or data store 52. While the specific
9 implementation of the PEMs 48, 50 will vary based on the available operating
10 system specific mechanisms available to intercept function calls and function call
11 returns relative to the operating system, the class of local domain accesses can be
12 described as intercepted by a local system PEM 48A, while the class of network
13 accesses are intercepted by a network PEM 48B.

14 [0045] On intercept of any interprocess communications request, whether
15 a local domain interprocess communications channel (IPC) or network socket
16 request, the requested access operation, along with authentication and
17 authorization information derived from the application instance process context
18 associated with the request, is reported to and processed through a rule-based
19 policy set maintained by the security appliance 42. Based on the request and
20 related information, an applicable set of policy rules are identified for evaluation
21 against the provided information. Access operations if and as permitted under
22 an applicable policy set are then enabled through the PEM to complete. Enabling
23 rules may qualify the access operation, such as to specify the establishment of an
24 encrypted communications channel through which the access operation is
25 permitted and whether encryption operations are to be performed locally by the

1 PEM or remotely through the security appliance 22. Where, similar to as shown
2 in Figure 1, multiple security appliances 22, 28 are assigned to the PEMs 48, 50,
3 communications between the security appliances 22, 28 permit mutual resolution
4 of access permissions under respectively identified policy sets.

5 [0046] In the particular case of a request to load a file for execution as an
6 application instance 56, a representative file load request is prepared and
7 forwarded by the PEM 48, 50 to the security appliance 42 for evaluation.
8 Preferably, a secure digital signature of the requested file is generated and
9 provided as part of the context authentication and authorization information
10 submitted to the security appliance 42. Although the requested file is typically
11 specified in an operating system call by a filesystem or UNC path, use of the
12 generated signature preferably provides a location independent identification of
13 the file upon which the determination to permit execution is based. In the
14 preferred embodiments of the present invention, the security appliance 42
15 maintains a pre-verified signature database for the executable files against which
16 policy determinations can be made. Based on the request data provided, the
17 security appliance 42 determines whether the file load request is permitted and
18 informs the PEM 48, 50 to either permit or deny the loading and execution of the
19 requested file.

20 [0047] In the case of requests to create or accept an IPC communications
21 session, the IPC session request and related context dependent information is
22 submitted to the security appliance 42 for evaluation. The response from the
23 security appliance 42 again determines whether the PEM 48, 50 enables the
24 requested communications channel. By considering separately the initial request
25 to create a channel and, on the target host computer system, the permission to

1 first to receive the request and then accept connection to the channel, the security
2 appliance 42 can evaluate the appropriateness of enabling the communications
3 session with respect to both the requesting source and target processes down to
4 the level of the individual source and target application instances and context
5 associated authorizations. Additionally, by intercepting both the creation and
6 acceptance of the communications channel session, the security appliance 42 can
7 coordinate the operation of the source and destination PEMs, typically PEMs 48,
8 50 in establishing a unique encrypted communications session channel.
9 Preferably, the security appliance 42 stores encryption keys defined through the
10 policy set rules as applicable to the source and target application instances and
11 operates to securely generate a session key unique for the particular
12 communications channel session established. In authorizing the creation of an
13 encrypted communications session, the session key is securely transmitted to the
14 PEMs 48, 50 and used to secure the communication channel for the duration of
15 the session.

16 [0048] A preferred architecture of a security appliance 60 is shown in
17 Figure 3. A Linux™-based appliance operating system 62 is preferably executed
18 on an Intel™ architecture hardware platform to support a dedicated control
19 program 64 that implements the security function of the security appliance 60.
20 One or more network interfaces 66_{1-N}, each managing the operation of an
21 underlying hardware network interface controller, provides connections to host
22 computer systems 12, 14 and other security appliances 28. Preferably,
23 communications between the PEMs 24, 26, 32 and other security appliances 28
24 are secured using a secure sockets layer (SSL) or similar secure network protocol.
25 Control connections transmitting request messages and responses can therefore

1 be routed variously through dedicated local networks as well as through shared
2 intranet and public networks. One or more dedicated cipher processors, such as
3 the HiFn™ 7986 security processor, are provided and controlled through cipher
4 processor interfaces 68_{1-N}. These cipher processors permit the security appliance
5 60 to perform appliance-based encryption and compression operations in support
6 of alternate deployment configurations of the security appliance 60.

7 [0049] A policy database 70 is provided locally on the security appliance
8 60 to store policy rule sets. A policy parser, implemented as a component of the
9 control program 64, executes to evaluate access requests as received by the
10 security processor 60 against matching policy rule sets. Operation of the control
11 program 64 and management of the policy database 70 are described in
12 Network Media Encryption Architecture and Methods for Secure Storage, by Pham
13 et al. (Application SN 10/016,897; filed December 3, 2001), which is
14 incorporated herein by reference. The policy parser preferably implements
15 decision tree logic to determine whether to allow a access request by matching
16 details of the request and associated context authentication and authorization
17 information against corresponding selectors of the policy rule sets. The type of the
18 request, whether classed as a program load, IPC operation, data file access, or
19 other, determines in part the relevant nature of the policy rule set selectors.
20 Preferably, the stored rules are specified by a system administrator to detail the
21 permitted operations against the various filesystem and communications resources
22 protected by the security processor 60 further qualified by applicable
23 authentication and authorization values and the time ranges within which a rule
24 is operative. The specified authorization values and time ranges are referred to
25 as the rule access attributes.

1 [0050] In the preferred embodiments of the present invention, the
2 authentication data provided in connection with a request processed through the
3 individual PEMs 24, 26, 32 is implicitly derived from the identifier of the process
4 that originates the request. Preferably, a secure identification of the user initiating
5 a particular request is established through use of a pluggable authentication
6 module (PAM) or similar operating system based application security module. In
7 accordance with the present invention, each PEM 24, 26, 32 intercepts the
8 operating system calls made to authenticate local users relative to a current
9 context processes. In particular, the return values for those calls are recorded by
10 the PEM 24, 26, 32. Preferably, on recognition of a successful authentication, the
11 local PEM 24, 26, 32 caches an authentication data record including at least the
12 authenticating process identifier. This authentication data may also record related
13 data including the type of authentication performed and details of the
14 authentication return values. Authentication attempts, including related process
15 context data, can be reported to and recorded by the associated security
16 appliances 60 for auditing and other administrative purposes.

17 [0051] Thus, for requests to be processed through to a security appliance
18 60, the process identifier associated with the request, as determined on intercept
19 by the local PEM 24, 26, 32 is used to retrieve a corresponding authentication
20 data record. The process identifier is used either directly or by tracing through the
21 chain of parent process identifiers maintained for the process context by the
22 operating system to match an authentication data record process identifier.
23 Where a context relevant authentication has not succeeded, a null authentication
24 data record is returned. The request to the security appliance 60 is then prepared
25 based on the contents of the authentication data record. The authentication data

1 preferably includes the request process identifier and, as applicable, the linking
2 parent process identifiers associated with the authentication data record. This
3 allows the subsequent qualification of the request on the basis of the type of
4 authentication performed and whether and to what extent inherited authentication
5 is acceptable.

6 [0052] Depending on the class type of the request, the access attributes
7 provided with a request can include the operation requested, the request source
8 host computer IP address, the request target host computer IP address, a target
9 resource identified by a path or other identifier, user identification, the source
10 application instance session and process identifiers, and a secure signature and
11 file size of the source application instance. The operative time of the request is
12 provided at least implicitly by the communication protocol used to transfer the
13 request to the security processor 60. Thus, for the class of file access requests, the
14 access attributes provided include the file operation requested, such as open,
15 read, write, append, delete, and move, and the applicable filesystem mount point,
16 path, and file specification. For communications oriented requests, the access
17 attributes provided will include the protocol type of the communication channel
18 requested, the source and target port numbers, and the network operation
19 request, such as open, read, write, and close. Thus, each request presented to the
20 security processor 60 is evaluated by the control program 64 against the
21 permissions matrix defined by the administratively defined policy rules to
22 determine whether the request is permitted. Depending on the determined policy
23 analysis result, a request response containing an enabled, qualified enable, or
24 denied status value is returned to the source PEM 24, 26, 32.

1 [0053] A signature database 72 locally provided on the security appliance
2 60 is also accessible to the control program 64. Preferably, the signature
3 database stores secure, SHA-1 based signatures for an administratively
4 determined set of executable programs, including associated executable library
5 files. A prototypical database, the National Software Reference Library (NSRL;
6 www.nsrl.nist.gov) which contains signatures for many conventional executable
7 programs, is available from the National Institute of Standards and Technology
8 (NIST). Preferably, as illustrated in Figure 4, the signature database 72 is
9 maintained as a content addressable list of signatures 82 against which individual
10 signatures can be matched. For the preferred embodiments of the present
11 invention, an intermediate reference data structure 84 is provided to permit
12 association of administratively selected sets of signatures into reference groups.
13 Each reference group is administratively identified by a unique resource identifier.
14 By administrative association, these resource identifiers can be referenced by the
15 resource access attributes of one or more potentially applicable policy rule sets
16 and thereby permit controlled determination of whether execution of the
17 corresponding signed executable is permitted.

18 [0054] The preferred procedure 90 of processing requests received by the
19 control program 64 is shown in Figure 5. Requests are received 92 variously from
20 the PEMs 24, 26, 32 and analyzed 94 to initially determine the class type of the
21 request as a program load 96, communications operation 98, data file access
22 100, or other request 102. For a program load request 96, the request provided
23 program signature is looked-up 104 against the signature list 82. A signature
24 look-up failure selects for a default program load policy. A successful look-up
25 104 identifies the signature as belonging to a reference group. The reference

1 group resource identifier and the authorization and access attributes provided with
2 the request 108 are then used to identify one or more matching policy rules 110.
3 The identified rules are evaluated 112, preferably in the reference group identified
4 order, to determine whether an enabled, conditional enabled, or denied response
5 message being returned 114 to the PEM 24, 26, 32 that originated the request.

6 [0055] The processing of communications requests 98, data access
7 requests 100, and other requests 102 is similar with the principle difference being
8 the request identification 98, 100, 102 and the authorization and access attributes
9 are used directly 108 as a selector of the applicable policy rule sets. Additionally,
10 where the result of the policy evaluation 112 is to enable the request, any ancillary
11 processing specified by the enabling policy rule set, such as to generate encryption
12 session tokens for establishing a secure communications channel, communicate
13 with other security appliances 22, 28, retrieve an encryption key for cipher
14 processing read/write data transfers, or retrieve compression parameters for use
15 in the processing of read/write data, is performed 116. Any applicable product
16 of the ancillary processing, such as encryption session tokens, is then returned
17 114 as part of the response message sent to the corresponding PEM 24, 26, 32.

18 [0056] In accordance with a preferred embodiment of the present
19 invention, data access requests 100 may involve additional request qualifying
20 data. For example, where the resource request identifies a read or write operation
21 directed against the Windows registry, the qualifying data 108 preferably includes
22 the target registry key, as derived by a PEM 24, 26, 32 relative to the operating
23 system call that would initiate the request. The registry key name, as well as the
24 request associated authentication and authorization data, is used to lookup 110
25 the applicable policy rules for evaluation 112. Again, the result of the policy

1 evaluation 112 is used to determine the content of the request response message
2 returned 114 by the control program 64.

3 [0057] The preferred system architecture 120 of a host computer or server
4 system 12, 14, 30 is shown in Figure 6. The hardware architecture is preferably
5 any conventional personal computer or workstation system including a host
6 processor 122, main memory 124, and network interface controller (NIC) 126.
7 A security coprocessor 128, supporting computationally intensive encryption and
8 compression operations, is optionally provided. An operating system 130, NIC
9 driver 132, native encryption and compression driver 134, and optional hardware
10 coprocessor driver 136 are executed within a kernel space 138, while program
11 instances, including application and operating system service instances 140, 142,
12 are executed in a user space 144 within the main memory 124.

13 [0058] In accordance with the present invention, a PEM 146 is locally
14 executed within the kernel space 138 as a component permitting interception of
15 selected application program interface (API) and virtual filesystem function calls
16 relative to the operating system 130. The specific mechanism for intercepting the
17 calls is operating system type and version dependent, though generally performed
18 by registering the PEM 146 with the kernel, where function intercepts are natively
19 supported or otherwise by redirection of the call entry points on initialization of the
20 PEM 146.

21 [0059] As shown in greater detail in Figure 7, a PEM 152 is preferably
22 installed as part of the operating system 130 logically architected as an operating
23 system interface PEM 152A, a network call intercept layer PEM 152B, and a
24 filesystem PEM 152C. The operating system interface and network call intercept
25 layer PEMs 152A, 152B are preferably used to qualify and control establishment

1 of local domain (domain socket, pipes, etc.) and network based (tcp, unix_socket,
2 etc.) communications channel sessions. The operating system interface PEM
3 152A, logically situated over the API call interface, can be further used to qualify
4 any call made to the operating system 130 including authentication calls. The
5 network PEM 152B is located in the logical call path between an application
6 instance 154 and a conventional network communications stack 156, including
7 a sockets layer 158. The file system PEM 152C operates to qualify file access
8 operations, including requests to load executable files and to access data and
9 other files.

10 [0060] As a component of the operating system 130, the operating system
11 kernel 160 is accessible by the operating system and network PEMs 152A, 152B
12 to determine the process context of the application instance 154, including the
13 authentication data and access attributes of both the specific process within which
14 the application instance 154 executes and any context associated parent
15 processes. For purposes of the present invention, a process context is defined as
16 a task parent process, such as a user login shell process or an operating system
17 service factory process, and the set of child processes traceable through parent
18 process identifiers to the task parent process, further related as inheriting the same
19 authentication and access attributes data as the task parent process. The
20 information describing the process context, as retrieved from the operating system
21 kernel 160, ultimately permits establishment of a communications channel
22 preferably specific to the application instance 154 or, alternately, to the member
23 processes of the process context that includes the application instance 154.

24 [0061] The filesystem PEM 152C is similarly implemented as an operating
25 system component to intercept filesystem related calls logically at the level of the

1 virtual filesystem switch (VFS) 162 or equivalent operating system structure. In a
2 preferred embodiment of the present invention, the filesystem PEM 152C utilizes
3 existing interfaces to permit logical insertion between the filesystem switch 162
4 and one or more conventional filesystems 164, such as the Microsoft® NTFS
5 filesystem, Unix® network filesystem (NFS), or Linux extended version two
6 filesystem (ext2). The operating system kernel 160 is also accessible by the
7 filesystem PEM 152C to determine the process context of the application instance
8 154 that originates a filesystem request directed to a local or network filesystem
9 164. Additionally, the filesystem PEM 152C provides for the generation of a
10 secure signature, preferably SHA-1 based, for any executable image loaded from
11 either a local or remote filesystem.

12 [0062] The PEM 152 communicates 166, as needed, with an assigned
13 security appliance 60 through the network stack 150 using either a shared
14 network interface 168 or a private network interface 170. By using the shared
15 network interface 168, the assigned security appliance 60 may be remotely
16 located on any connected intranet or public network accessible by the PEM 152
17 through the network stack 150. Thus, the PEM 152 may be implemented on a
18 host computer system geographically situated in a completely different location,
19 region, or country relative to the assigned security appliance 60, thereby allowing
20 the security appliance 60 to be physically secured while remotely protecting,
21 through strong encryption, any data accessible through the PEM 152 protected
22 host computer system, including direct attached storage local to the host computer
23 system. The PEM 152 can also be implemented in a notebook or other mobile
24 electronic device that directly or wirelessly connects to a network accessible
25 through the shared network interface 168.

1 [0063] Alternately, the private network interface 170, if provided, can be
2 used to connect one or more host computer systems with an assigned security
3 appliance 60 through a separate security network independent of any public or
4 even intranet-shared network. Use of a private security network permits the
5 connection to be made physically secure, enables use of alternate deployment
6 configurations particularly where clear text data is exchanged with the security
7 appliance 60, and ensures minimal latency in communications between a host
8 computer system and security appliance 60 by removing the albeit small
9 communications load between the PEM 152 and security appliance 60 from the
10 shared network 168 data path nominally used by the application instance 154.

11 [0064] In connection with distinguishing a permitted network-based
12 communications channel request, the assigned security appliance 60 performs the
13 ancillary processing necessary to provide a session specific encryption key to the
14 PEM 152. This session key is then utilized in operating system calls made from the
15 PEM 152 via a cipher driver interface 172 to, as appropriate, encrypt and decrypt
16 data in transit through the PEM 152. The cipher driver interface 172 interoperates
17 with the native encryption and compression driver 134 and hardware coprocessor
18 driver 136, if present, to manage the data processing preferably using the process
19 180 shown in Figure 8. In response to receiving data 182, typically inbound or
20 outbound with respect to the application instance 154, the presence of the
21 encryption coprocessor 128 is checked 184. In the absence, failure or queue full
22 state 186 of the encryption coprocessor 128, the received data is queued 188 for
23 native processing 190 through the native encryption and compression driver 134
24 using the host processor 122. Otherwise, the data is queued 192 for processing
25 194 by the encryption coprocessor 128, which is the preferred processing path.

1 The processed data is then routed 196 by the PEM 152, directly or indirectly to the
2 application instance 154, network stack 150, or filesystem 164.

3 [0065] Figures 9A, 9B, and 9C illustrated preferred system configurations
4 consistent with the present invention that provide for the secure binding of
5 application instances, through establishment of a secure communications tunnel
6 between securely identified process contexts. In Figure 9A, illustrating the
7 preferred configuration 200, a direct binding is established by requiring, through
8 operation of PEMs local to the host processes 202, 204, individual and mutual
9 qualification of the process contexts and the application instances, as executed
10 within the host processes 202, 204, by the assigned security appliances 206, 208.
11 In accordance with the present invention, the security appliances 206, 208 may
12 be a single device or two or more distinct physical devices that intercommunicate
13 as needed to coordinate consistent qualification operation with respect to the
14 process contexts including the host processes 202, 204. Individual qualification
15 of the host processes 202, 204 includes qualifying the creation of each the host
16 process 202, 204 for the execution of a securely identified application instance.
17 Mutual qualification includes qualifying the establishment of the encrypted tunnel
18 connection dependent on a combined consideration of the process contexts and
19 application instances. Where a session is qualified and specified by the qualifying
20 policy rule sets to be secure, an encrypted session key is generated by the security
21 appliances 206, 208 and provided to the respective PEMs to enable local
22 encryption operations 210, 212 to permit establishment of a direct, encrypted
23 communications channel.

24 [0066] Figure 9B shows an alternate configuration 220 where the secure
25 communications channel is established between the security appliances 206, 208,

1 preferably to offload the encryption and compression processing requirements of
2 the channel to the security appliances 206, 208. The PEMs locally executed
3 relative to the host processes 202, 204 qualify the participating process contexts
4 and application instances. The communications data, however, is transferred in
5 clear text or with conventional security encoding between the PEMs and the
6 security appliances 206, 208. Preferably, clear text links are made physically
7 secure.

8 [0067] The alternate configuration 230, shown in Figure 9C, as with the
9 configuration 220, utilizes a clear text link between the PEMs and security
10 appliances 206, 208 to permit utilization of the encryption and compression
11 processing capabilities of the security appliances 206, 208. Encrypted data is, in
12 this configuration 230, routed back through the PEMs to permit the encrypted
13 tunnel to be established directly between the securely identified process contexts.
14 In this manner, the presence and operation of the security appliances 206, 208
15 are hidden and the network data packets, as transmitted through the encrypted
16 communications channel are seen to originate from the routed through host
17 computer systems.

18 [0068] The preferred process 240 of securely qualifying an application
19 instance for execution is shown in Figure 10. On interception of an operating
20 system kernel 160 call, typically directed to the filesystem 164 to load a binary
21 image of a named program, the locally executed PEM 152 is invoked 242. The
22 authorization data and access attributes, including the execution target process
23 and process context, are determined from the operating system kernel 160. The
24 named program is then peremptorily loaded from the filesystem to permit
25 generation of a secure signature. Alternately, a program file access request is

1 submitted 244 to the assigned security appliance 60 to determine initially whether
2 program file is first accessible for loading in anticipation of execution. The access
3 request is either denied or the PEM 152 is enabled to load the requested program
4 file.

5 [0069] Once a program file is loaded from the filesystem, whether loaded
6 peremptorily or only subject to a successful access request, the program file is held
7 from execution by the PEM 152. A program execution request 246 is then
8 submitted to the assigned security appliance 60. This request preferably includes
9 the secure hash calculated signature of the program image and the authorization
10 data and access attributes determined by the PEM 152 for the program execution
11 request call context. Based on the request, the corresponding policy rule set is
12 evaluated to permit or deny execution of the program file. Where permitted, the
13 permission can be either express or conditional. Particularly in cases where
14 permission is conditional or denied, the ancillary policy implementation 116
15 preferably implements any administrative actions specified by the policy rule set,
16 which may include actions such as providing an alert message to an
17 administrative console, logging the request and associated data, issuing email
18 and pager notices of the event to administratively set addresses and numbers, and
19 generating execution qualifying control values to be returned to the PEM 152.

20 [0070] Thus, the response returned from the security appliance 60 includes
21 at least a binary value defining whether execution of the program file is to be
22 permitted or denied by the PEM 152. Where denied, the PEM 152 acting through
23 the operating system kernel 160, terminates the execution target process and
24 releases the program file image. Where permitted, the PEM 152 evaluates and
25 implements any conditional execution control values returned from the security

1 appliance 60. In a preferred embodiment of the present invention, these
2 conditional control values determine operative restrictions, such as execution time
3 period and priority, the issuance of local alert dialogs and logging levels, that are
4 then imposed on the execution context of the application instance by the PEM 152.
5 The loaded program file is then released to the operating system 160 to begin
6 execution 248 in the target context as the application instance.

7 [0071] The preferred process 250 of initiating of a secure communication
8 session between source and target program instances is shown in Figure 11.
9 While described relative to a network stack socket call to establish an
10 communications session between networked host computer systems, the process
11 250 is equally applicable to a communications session established through a
12 domain socket between processes executing on the same host computer system.
13 The request to create a network communications session is typically issued as a
14 network socket call from a program instance executed on the source host
15 computer system directed to the local socket layer 158. On functional interception
16 of the socket call 252 by the local PEM 152, the call specification is evaluated to
17 determine 254 the target host computer system and a specified port and transport
18 protocol. A connection request then is issued 256 from the source host computer
19 system to the assigned security appliance 60. This connection request preferably
20 includes the call specification data identifying the target host, port, and protocol
21 as well as data, as authentication data and access attributes acquired from the
22 local operating system kernel 160 including data identifying the source process
23 and process context. If the specific connection request is permitted under the
24 applicable policy rules, the PEM 152 is enabled to pass the socket call on to the

1 socket layer 158 to process and further relay a network call 258 to the specified
2 target host computer system.

3 [0072] On the specified target host computer system, the network call is
4 resolved, based on the port and protocol specification, to a communications
5 request directed to a specific program instance executing on the target host
6 computer system. The communications request is functionally intercepted by the
7 target executed PEM 152 and a corresponding session request is issued 260 to the
8 security appliance 60 assigned to the target host computer system. This session
9 request preferably includes the target process and process context related
10 authentication data and access attributes and an identification of the source host
11 computer system, port and protocol, as determined from the operating system
12 kernel 160. Preferably, a secure signature of the binary image of the target
13 program instance is also acquired and provided to the assigned security
14 appliance 60. Depending on the applicable policy rules, qualification of the
15 session request can be made dependent on any combination of the provided
16 session request information. The qualification can be further dependent on the
17 connection request information provided by the source host computer system. The
18 session request information is sufficient for a security appliance 60 assigned jointly
19 to the source and target host computer systems to determine the secure identity
20 of the source program instance. Where separate security appliances 60 are
21 assigned to the source and target host computer systems, the target assigned
22 security appliance 60 obtains sufficient information from the session request to
23 identify and, through secure interoperation, obtain the secure identity of the
24 source program instance from the source assigned security appliance 60.
25 Furthermore, the policy rules can consider other factors, such as time of day and

1 number of current connections established with the target program instance, to
2 finally determine whether the requested communication session is qualified and
3 therefore to be enabled.

4 [0073] Where a communication session is qualified, a session encryption
5 key is generated 262, either directly by a shared security appliance 60 or through
6 negotiation between source and target assigned security appliances 60. For
7 configurations where encryption processing is performed local to the source and
8 target host computer systems, the session key is then passed to the respective
9 PEMs 152. The communications request is then forwarded 264 from the target
10 PEM 152 to the target application instance to complete the initialization of the
11 secure communications session.

12 [0074] Once the communication session is established, protocol
13 appropriate commands and data can be transferred through the communications
14 tunnel represented by the communications session. These protocol commands
15 and data are fully encrypted while in transit between the source and target PEMs
16 152 utilizing the unique session key generated for the specific combination of
17 source and target applications instances. Thus, based on the generation of the
18 unique session tokens as specified by the applicable policy set rules, a secure
19 communications channel could be shared by multiple, similarly encoded sessions
20 or, preferably, each channel can host a uniquely encrypted communication
21 session securely bound specifically to the participating source and target
22 application instances.

23 [0075] Figure 12 shows the communication session process flows for
24 transmitting 270A and receiving 270B protocol commands and any associated
25 data, or equivalently protocol command responses and any associated data,

1 through a secure communication channel according to a preferred embodiment
2 of the present invention. From a source program instance, a protocol command
3 and any associated data, or data being returned in response to a command, is
4 functionally intercepted 274 by a PEM 152. By tracing the protocol call to the
5 source program instance, the applicable communications tunnel and session
6 information, including the process, process context and related data, is
7 determined 274 either directly from the local operating system kernel 160 or,
8 alternately, a local transient cache maintained by the PEM 152. This information,
9 combined with an identification of the specified protocol command or command
10 response, is provided 278 via the PEM 152 to the assigned security appliance 60
11 for qualification against the policy database 70. Specific protocol commands can
12 therefore be used as a basis for determining whether individual protocol
13 transactions within a communications session are permissible between specific,
14 securely bound program instances.

15 [0076] Where the command transaction is permitted, the security appliance
16 60 returns the session specific session key and, as may be appropriate for low-
17 bandwidth channels, any applicable compression control data. Any data being
18 transmitted is then optionally compressed 280 and the protocol command and
19 data are encrypted 282 using the session key. In accordance with the present
20 invention, each session key is held only transiently by the PEM 152 as necessary
21 for encrypting the corresponding protocol command and data. The encrypted
22 data is then transmitted 284.

23 [0077] On receipt 290, the PEM 152 determines the target process 292 for
24 the received encrypted data, and prepares a qualification request 294 including
25 an identification of the target process, process context and related data. The

1 session key and any applicable compression data are returned, permitting the
2 data to be decrypted 296 and decompressed as needed 298. The decrypted
3 protocol command and any applicable data are then provided to the target
4 program instance 300.

5 [0078] Finally, as generally shown in Figure 13, an existing communication
6 session can be terminated 310 and the corresponding secure communications
7 tunnel closed by any program instance closing the socket connection at either end
8 of the communications tunnel 312. Alternately, the communications session can
9 be terminated in response to an inactivity timeout 314 determined either by the
10 configuration of the network stack 150 or set and maintained by the PEMs 152
11 for each of the individual communications sessions managed through the PEMs
12 152. In each case, the secure communications tunnel is closed and any network
13 stack 150 and PEM 152 resources associated with the tunnel are released 316.

14 [0079] Thus, a system and methods for providing for establishing a secure
15 trust relationship between process contexts, down to the level of individual
16 program instances, has been described. While the present invention has been
17 described generally with reference to binary-based program instances, the present
18 invention is equally applicable to controlling the execution of byte-coded and
19 other encoding-based program instances.

20 [0080] In view of the above description of the preferred embodiments of the
21 present invention, many modifications and variations of the disclosed
22 embodiments will be readily appreciated by those of skill in the art. It is therefore
23 to be understood that, within the scope of the appended claims, the invention may
24 be practiced otherwise than as specifically described above.